Министерство науки и высшего образования и Российской Федерации

САНКТ-ПЕТЕРБУРГСКИЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

В. Э. Шмаков

СИСТЕМНЫЕ СРЕДСТВА LINUX И СЕТЕВАЯ БЕЗОПАСНОСТЬ

Учебное пособие

Санкт-Петербург
2024

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Операционная система (ОС) *Linux* является широко известным и типичным представителем класса открытых систем, имеет десятки разновидностей дистрибутивов, в том числе и отечественных, появившихся в результате успешного процесса импортозамещения. Данной ОС отводится все больше места в учебных планах университетов и колледжей. Особо важным компонентом учебного процесса являются лабораторные занятия, где студенты имеют возможность получить практические навыки работы, администрирования и разработки ПО под *Linux,* с использованием сетевых и других системных возможностей этой ОС.

Тематика лабораторных работ тесно переплетается с содержанием основного курса лекций, а примеры многих программ рассматриваются в лекциях, и их исходные файлы имеют такие же имена, как указаны в заданиях лабораторных работ. Поэтому изложение теоретического материала в описании лабораторных работ минимизировано, чтобы не дублировать лекции. В качестве основной литературы рекомендуется, в первую очередь, использовать конспект лекций и, кроме того, в изданиях [1 – 6] можно найти много полезной информации.

Ресурсы Интернета в области Linux систем и сетевых технологий поистине неисчерпаемы. В данной работе местами приводятся ссылки на конкретные ресурсы, но это не значит, что следует ограничиваться лишь ими. Каждый сможет выбрать себе в поисковиках такие ресурсы, которые будут в наибольшей степени отвечать его индивидуальным запросам и предпочтениям в манере представления материала.

ЛАБОРАТОРНАЯ РАБОТА № 1

Базовые команды ОС Linux

Цель работы. Освоение минимального набора базовых команд операционной системы *Linux*, знакомство с файловой системой, особенностями прав доступа, получение первичных навыков работы под *Linux*.

Последовательность выполнения работы:

1. Запустите терминал и выполните серию команд *shell*, рассмотренных в материалах лекций. Такие, как *pwd* , *ls* , *cd* , *mkdir* , *rm* . Полное описание синтаксиса и семантики этих и любых других команд можно увидеть в системе помощи ОС *Linux*, вызываемой с терминала в виде:
man < интересующая вас команда >. Или же запускайте веб-браузер и используйте всю информационную мощь Интернета.

2. Проанализируйте результаты выполнения команд. Наиболее значимые скриншоты (снимаются нажатием клавиш *Alt + Prnt Scrn*) поместите в отчет.

3. Создайте дерево каталогов глубиной вложения от трех уровней, а в самих каталогах создайте текстовые файлы. Примените различные способы создания новых файлов из командной строки.

4. Запустите с терминала *Midnight Commander* вводом команды *mc* и ознакомьтесь с его основными возможностями по работе с файловой системой. Наполните созданные на предыдущем шаге файлы каким-либо содержанием. Для этого можно использовать любой редактор, от *vi* , встроенного в ОС, до графического редактора *gedit* , вызываемого из графической оболочки ОС.

5. Выполните на терминале вторую серию команд *cat* , *cp* , *find* , *link* , *chmod* , рассмотренных в лекциях. Для манипуляций с помощью этих команд используйте текстовые файлы, созданные и наполненные на предыдущем шаге.

6. Попытайтесь создать на своем дереве какой-нибудь каталог с правами доступа, аналогичными каталогу *darkroom* , рассмотренному в лекциях.

ЛАБОРАТОРНАЯ РАБОТА № 2

Запуск и завершение процессов

Цель работы. Знакомство с характерной для *Linux* схемой порождения и завершения процессов, с отношениями типа потомок – родитель, со способами передачи информации о событии завершения процесса.

В данной и в последующих работах понадобится компилировать исходные файлы с помощью строчного компилятора *g++* . В простейших случаях одномодульных проектов применяется команда *g++ < имя.cpp файла >* , исполняемый файл генерируется только в случае отсутствия ошибок компиляции и по умолчанию именуется *a.out.* Запускается исполняемый файл с терминала с указанием полного пути (например, */a.out*, если он в текущем каталоге) или из *Midnight Commander-a.* Сведения о полной функциональности *g++* можно почерпнуть в справке *man* .

Последовательность выполнения работы:

1. Скомпилируйте, выполните (и поясните в отчете получающийся вывод на консоль) примеры программ *forkdemo.cpp* , *tinymenu.cpp* , *tinyexit.cpp* , *procgroup.cpp* , *wait_parent.cpp* . Процесс *wait_parent* при исполнении запускает процесс *wait_child* . Программа *wait_child.cpp* компилируется с опцией: *g++ wait_child.cpp-o wait_child* . Описание данных программ можно найти в тексте лекций. При необходимости конвертации текстовых файлов из формата *DOS* в *Linux*, и наоборот, используйте команды *dos2unix* и *unix2dos* .

2. Модифицируйте программу *forkdemo.cpp* (или создайте собственную), так чтобы ввод/вывод на терминал отсутствовал, а при проходе по циклу была временная задержка, например, *sleep()*. Запустите эту программу в фоновом режиме (*background*), введя при запуске символ *&* после пробела и зафиксировав значение *PID*, назначенное системой фоновому процессу при запуске. Выполните на терминале команды *ps* , *top* , *uptime* , *pstree* . Снимите свой фоновый процесс командой *kill* с соответствующими параметрами.

Скриншоты вместе с пояснениями к выполнению процессов и команд, а также исходные тексты программ, составленных вами самостоятельно, приведите в отчете.

3. Проанализируйте работу системного вызова *wait() на* примере предоставленном в исходных файлах *wait_parent cpp* и *wait_child cpp* . Какую информацию процесс-родитель может получить о завершении потомка?

4. Исследуйте, что произойдет, если процесс-потомок сменит текущий каталог, будет ли изменен текущий каталог для родителя? Создайте программу, подтверждающую ответ. При запуске сначала ваш процесс должен выдать на консоль информацию о текущем каталоге. Затем должен быть запущен процесс-потомок, который должен изменить текущий каталог на другой и также выдать информацию о нем на консоль. После этого потомок должен завершиться с возвратом управления родителю, который, в свою очередь, должен снова выдать на консоль информацию о текущем каталоге в момент после завершения потомка.

5. Проиллюстрируйте как процесс-родитель и процесс-потомок разделяют один и тот же дескриптор и смещение текстового файла. Для этого составьте программу, в которой процесс-родитель должен открывать текстовый файл и читать порцию данных, а затем запускать потомка. Потомок должен читать еще порцию данных из открытого файла и выводить на консоль. По завершению потомка родитель должен продолжать чтение из того же файла и выводить результат на консоль. Можете использовать вызов *sleep()* для синхронизации доступа родителя и потомка к файлу.

ЛАБОРАТОРНАЯ РАБОТА № 3

Программные каналы

Цель работы. Изучение конвейеров (*pipes*, программных каналов), как простейшего средства коммуникации запущенных процессов. Исследование способов организации каналов различных видов и их сопоставление.

Последовательность выполнения работы:

1. Скомпилируйте и выполните программу *whosortpipe.cpp* . Сопоставьте результат выполнения программы с выполнением этих же двух команд с терминала в конвейерном режиме (с использованием < | >). Анализ результатов работы этой программы (как и всех последующих) с соответствующими скриншотами приведите в отчете.

2. Программу *cmdpipe.cpp* запускайте после компиляции, задавая ей при стартах в качестве параметров командной строки пары команд *shell* для конвейеризации (*who* и *sort* ; *last* и *sort* ; *last* и *more* ; *pstree* и *more*). Сопоставьте результаты запусков программы с выполнением тех же пар команд из *shell* в конвейерном режиме.

3. Создайте программный канал типа *named pipe* из командной строки (пример в лекциях). Используя необходимые системные вызовы организуйте канал *named pipe* в программе, сравните результат выполнения обмена по нему в программе с тем, чего можно достичь, создавая *named pipe* на терминале.

4. Разберите и выполните пример клиент-серверного взаимодействия, организованного на конвейерах различного типа. Исходный текст примера содержится в файлах *pipe_server.cpp* , *pipe_client.cpp* и *pipe_local.h* и разобран в материалах лекций. Север запускается в фоновом режиме. Проанализируйте какие конвейеры используются, как они создаются, как функционирует данная системы, ее недостатки. Программа-сервер этого примера исполняет каждый командный запрос поочередно. Если какой-либо запрос потребует много времени, все остальные клиентские процессы будут ожидать обслуживания.

5. Модифицируйте программу *pipe_server.cpp* из предыдущего задания так, чтобы при получении нового сообщения от очередного клиента сервер порождал очередной дочерний процесс для выполнения задачи обслуживания данного запроса (исполнения переданной от клиента команды и переправки результата обратно клиенту).

ЛАБОРАТОРНАЯ РАБОТА № 4

BASH. Командные файлы. Переменные окружения

Цель работы. Знакомство с важным атрибутом любой операционной системы – переменными среды (или переменными окружения) *environment variables* и с возможностями их использования в *Linux*.

Освоение языка для составления командных сценариев и написание набора полезных для системного администрирования скриптов.

Последовательность выполнения работы:

1. Создайте несколько символьных переменных среды (переменных окружения). Составьте командный файл (*bash* сценарий*)*, выводящий на консоль значения этих переменных. Выполните операцию конкатенации (склеивания) значений переменных и выведите полученный результат на консоль. Выделите из конкатенированной переменной среды подстроку и выведите ее на консоль. Замените выделенную подстроку на какое-либо другое значение и выведите измененное значение переменной среды на консоль.

2. Создайте несколько переменных среды в интерпретации, как числовые переменные. В новом командном файле выполните с этими числовыми переменными все допустимые арифметические операции, выводя на консоль результаты операций и соответствующие комментарии.

3. Создайте командный файл (основной), выдающий при старте сообщение и затем вызывающий другой командный файл (имя этого другого файла задается при старте основного файла в качестве параметра командной строки), который выдает свое сообщение и затем приостанавливается до нажатия любой клавиши. При возврате управления в вызывающий (основной) командный файл из него должно выдаваться еще одно сообщение, подтверждающее возврат управления.

4. Составьте командный файл, выводящий на экран различия содержимого двух каталогов, имена которых передаются в качестве параметров. Отличия следует искать в именах файлов, их размерах и атрибутах.

5. Разработайте командный файл сценария для поиска текстовых

файлов, содержащих заданную последовательность символов. Эта последовательность передается при запуске в качестве первого параметра командной строки. В качестве второго параметра передается имя файла результатов, который должен быть создан в сценарии для записи в него имен найденных текстовых файлов и номеров их строк, в которых содержится заданная последовательность символов.

ЛАБОРАТОРНАЯ РАБОТА № 5

Учетные аписи. Foreground, background

Цель работы. Манипуляции с правами доступа при создании в системе учетных записей и исследование влияния прав на файловые операции. Изучение специфик фонового (*background*) и диалогового (*foreground*) режимов исполнения процессов и способов переключений между этими режимами.

Последовательность выполнения работы:

1. Создайте учетные записи для нескольких пользователей (не задавая им прав администратора) и объедините их в две группы. Заходя в систему под разными аккаунтами, создайте в соответствующих домашних каталогах файлы, варьируя при этом права доступа для пользователя, для группы, для всех. Убедитесь, что права доступа разделяются в соответствии с тем, как это задано. Проведите операцию слияния файлов с различными правами доступа и проверьте, какие при этом получаются права у результирующего файла.

2. Запустите в фоновом (*background*) режиме командный файл (процесс), выдающий в цикле с некоторой задержкой сообщение на консоль. Запустите другой командный файл (процесс), требующий диалога, в обычном режиме (*foreground*). Убедитесь в том, что вывод этих двух процессов на консоль перемежается. Остановите фоновый процесс сигналом *kill*. Запустите его снова, организовав предварительно перенаправление его вывода в файл. Убедитесь, что теперь вывод двух процессов разделен.

3. Доработайте предыдущее задание так, чтобы показать возможность

перевода фонового процесса в диалоговый режим (*foreground*) для выполнения операции ввода с клавиатуры и, затем, возврата его обратно в фоновый (*background*) режим (команды *fg , bg , jobs*).

4. Разработайте командный файл для выполнения архивации каталога через определенные интервалы времени. Запустите командный файл в режиме *background.* Имя архивируемого каталога, местоположение архива и время (период) архивации передаются при запуске командного файла в виде параметров командной строки.

5. Создайте командный файл, который синхронизирует содержимое заданного каталога с эталонным. После запуска и отработки командного файла в заданном каталоге должен оказаться тот же набор файлов, что и в эталонном (если файла нет – он копируется из эталонного каталога, если найдется файл, которого нет в эталонном, – удаляется). Если файл с некоторым именем есть и в заданном и в эталонном каталогах, то он перезаписывается только в том случае, если в эталонном имеется более новая версия файла. Имена обоих каталогов должны передаваться командному файлу при старте параметрами командной строки.

ЛАБОРАТОРНАЯ РАБОТА № 6

Генерация и обработка сигналов

Цель работы. Освоение простейшего средства управления процессами, позволяющего процессам передавать информацию о каких-либо событиях, отрабатывать реакции на различные события и взаимодействовать друг с другом.

Последовательность выполнения работы:

1. Программа *sigint.cpp* осуществляет ввод символов со стандартного ввода. Скомпилируйте и запустите программу и отправьте ей сигналы *SIGINT* (нажатием *Ctrl-C*) и *SIGQUIT* (нажатием *Ctrl-*). Проанализируйте результаты.

2. Запустите программу *signal_catch.cpp* , выполняющую вывод на консоль. Отправьте процессу сигналы *SIGINT* и *SIGQUIT* , а также *SIGSTOP*

(нажатием *Ctrl-Z*) и *SIGCONT* (нажатием *Ctrl-Q*) . Проанализируйте поведение процесса и вывод на консоль, а также сравните с программой из предыдущего пункта.

3. Скомпилируйте и запустите программу *sigusr.cpp* . Программа выводит на консоль значение ее PID и зацикливается, ожидая получения сигнала. Запустите второй терминал и, отправляя с него командой *kill* различные сигналы, в том числе и *SIGUSR1* , проанализируйте реакцию на них.

4. Составьте программу, запускающую процесс-потомок. Процесс-родитель и процесс-потомок должны генерировать (можно случайным образом) и отправлять друг другу сигналы (например *SIGUSR1, SIGUSR2*). Каждый из процессов должен выводить на консоль информацию об отправленном и о полученном сигналах.

5. Для организации обработчиков сигналов предпочтительно использовать системный вызов *sigaction()* и сответствующую структуру данных. Обеспечьте корректное завершение процессов.

6. Модифицируйте программу *занятия 3* (файлы *pipe_server.cpp*, *pipe_client.cpp* и *pipe_local.h*) сделав ее более стабильной в работе. В числе недостатков, которые желательно устранить, можно, например, указать:

– если клиентский процесс завершается по получению сигнала *SIGINT* (*Ctrl+C),* то *private FIFO* не удаляется из системы (исправляется посредством организации перехвата сигнала с выполнением необходимых действий);

– клиентский процесс при его инициализации может обрушиться, если сервер окажется недоступен (исправляется путем попытки запуска сервера из клиента, если сервер не активен).

ЛАБОРАТОРНАЯ РАБОТА № 7

Семафоры и синхронизация

Цель работы. Освоение семафоров (semaphores) как эффективного средства синхронизации доступа процессов к разделяемым ресурсам операционной системы.

Последовательность выполнения работы:

1. Скомпилируйте и выполните программу *gener_sem.cpp*, иллюстрирующую создание наборов с семафорами или получение доступа к ним. Запустите программу несколько раз и после каждого ее завершения выполните команду *ipcs -s* .

Поясните зависимость процедуры создания семафоров от используемых в вызове *semget()* флагов и их комбинации.

2. Удалите созданные на предыдущем шаге семафоры с помощью команды *ipcrm* с соответствующей опцией и значением *id* семафора или ключа.

3. Скомпилируйте s*emdemo.cpp*, демонстрирующую организацию разделения доступа к общему ресурсу между несколькими процессами с помощью технологии семафоров. Запустите сразу несколько процессов на разных терминалах и проанализируйте их взаимодействие и соблюдение очередности в попытках получения общего ресурса.

4. Скомпилируйте программу *semrm.cpp* и произведите с ее помощью удаление созданного на предыдущем шаге семафора. Поясните, почему данная программа удаляет только те семафоры, которые были созданы при выполнении программы *semdemo.cpp* . Попробуйте удалить семафор с помощью запуска *semrm.cpp* во время исполнения s*emdemo.cpp* и проанализируйте ситуацию.

5. Модифицируйте программу s*emdemo.cpp* , например, предоставив процессу возможность после освобождения ресурса становиться снова в очередь на повторное его занятие (а не завершаться), организовав, при этом, завершение процесса по вводу какого-либо символа.

6. Составьте программу, позволяющую мониторить количество процессов (типа s*emdemo*), находящихся в состоянии ожидания освобождения ресурса (Trying to lock...) в каждый момент времени.

Программа строится на основе вызова *semctl()* с соответствующими параметрами и запускается на отдельном терминале.

ЛАБОРАТОРНАЯ РАБОТА № 8

Обмен через очереди сообщений

Цель работы. Знакомство с возможностями очередей сообщений (Message Queues) – мощного и гибкого средства межпроцессного взаимодействия в ОС *Linux.*

Последовательность выполнения работы:

1. Скомпилируйте и выполните программу *gener_mq.cpp* , создающую несколько очередей сообщений. После завершения программы выполните команду *ipcs* и поясните отличие результата от того, что был при вызове подобной команды из программы.

Поясните зависимость процедуры создания очереди сообщений от используемых в вызове *mqmget()* флагов и их комбинации.

2. Скомпилируйте программы *sender.cpp* и *receiver.cpp* , задав соответствующим исполняемым файлам разные имена: (*g++ < имя .cpp файла > -o < имя .out файла >*). Запустите процессы на разных терминалах и передайте текстовые сообщения от процесса *sender* процессу *receiver*. Проанализируйте, что происходит с ресурсом *Message Queue* после завершения каждого из процессов (командой *ipcs*). При этом выполните различные виды завершения отправкой сигналов *SIGQUIT* и *SIGINT* (нажатием *Ctrl-C*) .

3. Что происходит, если процесс *receiver* запускается уже после того, как процесс *sender* отправил в очередь одно или множество сообщений, или запускается повторно после завершения?

4. Запустите несколько процессов *receiver* на различных терминалах и, отправляя сообщения процессом *sender* , проанализируйте ситуацию.

5. Модифицируйте программы *sender.cpp* и *receiver.cpp* так, чтобы организовать отправку сообщений, по крайней мере, двух типов через одну и ту же очередь для нескольких различных процессов получателей. Для этого необходимо управлять параметром в поле *mtype* структуры *my_msgbuf* на передающей стороне и параметром *msgtyp* в системном вызове *msgrcv()* на приемной стороне.

ЛАБОРАТОРНАЯ РАБОТА № 9

Работа с разделяемой памятью

Цель работы. Использование для обмена данными разделяемой памяти (*shared memory*) – самого быстрого средства межпроцессного взаимодействия в *Linux*.

Последовательность выполнения работы:

1. Скомпилируйте и выполните программу *gener_shm.cpp* демонстрирующую создание сегментов разделяемой памяти. Запустите программу несколько раз и после каждого ее завершения выполните команду *ipcs -m* . Поясните зависимость процедуры создания сегментов разделяемой памяти от используемых в вызове *shmget()* флагов или их комбинации.

2. Удалите созданные на предыдущем шаге сегменты разделяемой памяти с помощью команды *ipcrm* с соответствующей опцией и значением *id* сегмента или ключа.

3. Скомпилируйте s*hmdemo.cpp*, осуществляющую операции записи в разделяемую память без разделения доступа к этому общему ресурсу. Символы, записываемые в общую память, передаются в качестве параметра командной строки при запуске процесса s*hmdemo*. Запуск этого процесса без параметров приводит к выводу на консоль текущего содержимого сегмента общей памяти.

4. Запустите несколько раз процессы типа s*hmdemo* с различными значениями параметров и проиллюстрируйте возможности чтения и записи в сегмент общей памяти независимо исполняемыми процессами. Затем удалите сегмент памяти командой *ipcrm* .

5. Скомпилируйте и выполните программу *attach_shm.cpp* , иллюстрирующую передачу символьной информации между двумя процессами (родственными) через сегмент общей памяти с модификацией этой информации. Проанализируйте значения выводимой информации о границах сегментов в системной памяти. За счет чего после завершения данной программы сегмент общей памяти уже не присутствует в системе?

Модифицируйте приложение так, чтобы присоединение сегмента разделяемой памяти происходило по некоторому указанному конкретно адресу, а не на усмотрение самой операционной системы.

6. Составьте программу, создающую три разделяемых сегмента памяти размером 1023 байта каждый. Укажите в вызове *shmat()* параметр *shmaddr = 0* при привязке сегментов. Разместит ли операционная система сегменты в последовательных участках? Позволит ли система ссылку или изменение 1024-го байта любого из этих участков?

7. Выполните пример синхронизации обмена на семафорах из лекции по *shared memory*. Исходники для указанного примера предоставлены набором файлов *local.h* , *parent.cpp* , *producer.cpp* , *consumer.cpp* .

ЛАБОРАТОРНАЯ РАБОТА № 10

Создание соединений на Socket L4 level

Цель работы. Освоение набора системных вызовов для создания сокетных соединений различных типов для обмена данными по сети.

Последовательность выполнения работы:

1. Скомпилируйте и выполните программу *socketpair.cpp* , иллюстрирующую создание сокета упрощенного вида и обмен данными двух родственных процессов. Проанализируйте вывод на консоль. Существует ли зависимость обмена от различных соотношений величин временных задержек (в вызовах *sleep()*) в процессе родителе и в процессе потомке?

2. Скомпилируйте программы *echo_server.cpp* и *echo_client.cpp* , задавая им при компиляции разные имена (размещаем файлы в одном каталоге). Запустите программы сервера и клиента на разных терминалах. Вводите символьную информацию в окне клиента и проанализируйте вывод. Какому типу принадлежат сокеты, используемые в данном примере клиент-серверного взаимодействия? С чем связано создание специального файла в текущем каталоге во время исполнения программ?

3. Скомпилируйте с разными именами программы *sock_c_i_srv.cpp* и

sock_c_i_clt.cpp (в них используется общий *include* файл *local_c_i.h*). Запустите программы сервера и клиента на разных терминалах. При запуске клиента указывайте в качестве параметра командной строки имя хоста *localhost*. Вводите символьную информацию в окне клиента и поясните вывод. Какому типу принадлежат сокеты, используемые в данном примере клиент-серверного взаимодействия?

4. Модифицируйте программу *echo_server.cpp* так, чтобы при ответе на запросы клиента что-либо выводилось в окне сервера. Испытайте работу эхо-сервера при одновременной работе с несколькими клиентами.

ЛАБОРАТОРНАЯ РАБОТА № 11

Взаимодействие процессов по сети

Цель работы. Создание клиент-серверных приложений, взаимодействующих друг с другом по сети на основе технологии соединения на сокетах.

Последовательность выполнения работы:

1. Скомпилируйте и запустите программу *server_game.cpp* , иллюстрирующую обмен данными с клиентскими приложениями по итеративной схеме.

2. Запустите другой терминал и проверьте с него наличие в системе созданного сервером сокета и то, что он находится в состоянии *LISTEN*. Для этого выполните команду *netstat -a | grep 1066* . Проанализируйте вывод данной команды и объясните ее смысл.

3. Запустите в качестве клиентского процесса утилиту *telnet localhost 1066* . При организации коммуникации по сети, на разных компьютерах вместо *localhost* при запуске клиента указывается *IP-адрес* компьютера, на котором был запущен сервер.

4. Диалог с сервером заключается в угадывании слова. Оно вводится по буквам с клиентского терминала. При этом сервер вместо неугаданных букв выдает символы ”-” , а также считает число оставшихся неудачных попыток (всего их предусмотрено 12).

5. Завершите серверное приложение с помощью сигнала *kill* , и затем определите командой *netstat -a | grep 1066* когда исчезает из системы соединение на сокетах. Во время сеанса обмена также примените команду *netstat -a | grep 1066* , чтобы исследовать состояние соединения.

6. Проделайте все заново, но запускайте не одно клиентское приложение (в виде *telnet*), а несколько экземпляров с разных терминалов, и попытайтесь работать с них одновременно. Проанализируйте, как сервер будет обслуживать запросы в этом случае.

7. Модифицируйте программу *server_game.cpp* так, чтобы запросы от каждого из клиентов могли обслуживаться конкурентно (путем запуска для каждого нового соединения собственного нового процесса на сервере. Возможно также улучшить качество самой игровой функции *guess_word()* сервера. Проанализируйте, как обслуживаются запросы в случае конкурентной схемы работы сервера.

ЛАБОРАТОРНАЯ РАБОТА № 12

Анализ сетевого трафика. Wireshark

Цель работы. Получение навыков работы с мощным инструментом фильтрации трафика и обнаружения вторжений, со снифером *Wireshark.*

Последовательность выполнения работы:

1. Установите и запустите в привилегированном режиме анализатор сетевого трафика *Wireshark*. О базовой функциональности снифера *Wireshark* можно узнать, например, из учебных роликов, выложенных на ресурсах:

http://www.youtube.com/watch?v=6X5TwvGXHP0
http://www.youtube.com/watch?v=r0l_54thSYU
http://www.youtube.com/watch?v=qs_DqMdlKHY

2. Отфильтруйте трафик протокола *ICMP* (трафик порождается, например, утилитами *ping, traceroute*). Приведите в отчете подробный формат пакета, содержащего *ICMP-сообщение* с пояснением назначения каждого из полей.

Воспроизведите различные режимы работы утилит и приведите снятые снифером дампы пакетов с соответствующими этим режимам кодами сообщений или ошибок в полях пакетов.

3. Проанализируйте трафик *ARP* (протокола преобразования адресов). поясните предназначение *ARP-таблиц* и приведите (с пояснениями) дампы *ARP-сообщений*, снятые снифером.

4. Установите на компьютере *FTP-сервер* или воспользуйтесь одним из имеющихся в открытом доступе.

5. В классической версии протокола *FTP* имена и пароли пользователей передаются по незащищенным сетям в открытом виде. Продемонстрируйте уязвимость протокола *FTP* путем извлечения из пакетов, с помощью анализатора трафика, информации об именах и паролях пользователей на этапе аутентификации пользователя.

6. Выполните, по возможности, настройки, повышающие уровень защиты *FTP-сервера* (измените текст приветствия, организуйте отправку баннеров соединений, обезопасьте анонимный доступ), и проверьте работу настроек *FTP-сервера*, соединяясь с ним с клиентского приложения.

7. Сопоставьте защищенность протоколов удаленного доступа *Telnet* и *SSH*. Действуйте по схеме, аналогичной демонстрации уязвимости протокола *FTP*.

8. Проанализируйте с помощью сетевого снифера Wireshark сообщения транспортного уровня: UDP-дейтаграммы и TCP-сегменты. Прокомментируйте содержимое полей заголовков перехваченных дейтаграмм и сегментов.

ЛАБОРАТОРНАЯ РАБОТА № 13

Фильтрация трафика. Правила IPtables.

Цель работы. Освоение технологии *iptables,* составление правил фильтрации сетевого трафика, выделение отдельных сетевых протоколов, *IP-адресов*, номеров портов. Создание простейших программных сетевых экранов *(firewalls).*

Последовательность выполнения работы:

1. Войдите в систему и определите *IP-адрес* вашего компьютера. В дальнейшем также понадобится пароль администратора для вашего компьютера (узнайте у преподавателя).

2. Просмотрите текущие правила, установленные в *iptables*. Результаты выполнения ваших команд, а также последствия применения вводимых вами правил протоколируйте в отчете о лабораторной работе. Никогда не сохраняйте вводимые правила командами типа *save*.

3. Введите правило, блокирующее весь входящий трафик на Ваш компьютер. Проверьте наличие введенного правила в системе.

4. Добавьте правила для фильтрации входящего трафика (с использованием целей *DROP* и *ACCEPT*) так, чтобы веб-трафик проходил, а остальной был блокирован. Просмотрите детализированный список правил и проверьте их действие.

5. Закройте вход *ICMP-пакетам*, отправляемым с какого-либо другого компьютера лаборатории. Создав соответствующее правило (испытайте цели *DROP* и *REJECT*), проверьте его действие утилитой *ping* .

6. Убедитесь, что при использовании цели *REJECT*, в отличие от *DROP*, выдается сообщение об ошибке на машину отправитель.

7. Организуйте на Вашем компьютере форвардинг с какого-либо порта внешнего интерфейса на определенный порт другой машины .

8. Проиллюстрируйте применение действия *MIRROR* таблицы *NAT*, используемое для защиты от сканирования портов.

Для углубленного изучения *iptables* подойдет первоисточник, с наиболее подробным описанием:

https://www.opennet.ru/docs/RUS/iptables/#ACCEPTTARGET

https://www.frozentux.net/iptables-tutorial/iptables-tutorial.html

и другие.

ЛАБОРАТОРНАЯ РАБОТА № 14

Сетевое экранирование.

Цель работы. Составление правил фильтрации, создание простейших сетевых экранов, журналирование передаваемых по сети пакетов, дифференцирование логов по разным событиям.

Последовательность выполнения работы:

1. Для целей безопасности и сокрытия внутренней сетевой инфраструктуры *ICMP ping* запросы извне можно запретить :

sudo iptables -A INPUT -p icmp --icmp-type echo-request -j DROP

2. Чтобы открыть хост для входящих *ICMP* запросов и исходящих ICMP ответов выполняются правила:

sudo iptables -I INPUT -i eth0 -p icmp --icmp-type 8 -s 0/0 -d

$SERVER_IP -m state --state NEW, ESTABLISHED, RELATED -j ACCEPT

sudo iptables -I OUTPUT -i eth0 -p icmp --icmp-type 0 -s

$SERVER_IP --d 0/0 -m state –state ESTABLISHED, RELATED -j ACCEPT

где: *$SERVER_IP - IP*-адрес хоста,

icmp-type 8 - эхо-запрос,

icmp-type 0 - эхо-ответ,

0/0 - любой адрес.

3. Ограничить ping запросы определенным количеством в единицу времени можно:

sudo iptables -A INPUT -p icmp -m icmp -m limit -limit 1/second -j ACCEPT

4. Заблокировать все входящие запросы с определенного адреса (например, *192.168.0.6*) можно:

sudo iptables -A INPUT -s 192.168.0.6 -j DROP

5. Чтобы не пропускать все входящие запросы порта *80*, надо выполнить:

sudo iptables -A INPUT -p tcp --dport 80 -j DROP

6. Если необходимо заблокировать входящий запрос порта *80* с определенного адреса (например, *192.168.0.6*), тогда:

sudo iptables -A INPUT -p tcp -s 192.168.0.6 --dport 80 -j DROP

7. Сбросить трафик с определенного *(00:0F:EA:91:04:08) MAC* адреса можно:

sudo iptables -A INPUT -m mac –mac-source 00:0F:EA:91:04:08 -j DROP

или разрешить только для протокола *TCP:*

sudo iptables -A INPUT -p tcp --destination-port 22 -m mac –mac-source 00:0F:EA:91:04:08 -j ACCEPT

8. Составьте правило логирования пингов и правило разрешающее пинги:

iptables -A INPUT -p ICMP --icmp-type 8 -j LOG --log-prefix "Ping detected:"

iptables -A INPUT -p ICMP --icmp-type 8 -j ACCEPT

Правила с действием *LOG* изложены, например, на ресурсе

http://www.opennet.ru/docs/RUS/iptables/#LOGTARGET

9. Организуйте журналирование пакетов так, чтобы обеспечить запись в отдельный файл и не дублировать сообщения в системные логи. Создайте правила для разных событий, при этом, каждое событие направляйте в свой лог. Попытайтесь настроить ротацию логов *iptables* и убедитесь в ее правильности.

ЛАБОРАТОРНАЯ РАБОТА № 15

Интерфейс nftables .

Цель работы. Выполнение задач фильтрации трафика, форвардинга, логирования и др. с помощью *nf_tables* с использованием утилиты *nft*, функционально схожей с *iptables*, но более развитой и предоставляющей удобный синтаксис.

Последовательность выполнения работы:

1. Из описания работ 13 и 14 выберите несколько задач (любых, около 4-х , 6-ти) и выполните их с помощью *nftables*.

Сопоставьте полученные результаты, сделайте выводы об удобстве и трудоемкости применения обеих технологий (*iptables* и *nftables*).

Знакомство с технологией *nftables* можно начать, например, по предоставленным материалам, по видеороликам :

Nftables ЧАСТЬ 1 межсетевой экран

https://www.youtube.com/watch?v=hUkZC9snX2A

Nftables ЧАСТЬ 2

https://www.youtube.com/watch?v=M37v3KD0qDA

Nftables ЧАСТЬ 3

https://www.youtube.com/watch?v=9E5qPe3bg18

Nftables HOWTO

https://wiki.nftables.org/wiki-nftables/index.php/Main_Page

Netfilter project

http://nftables.org/documentation/index.html

Debian man nft

https://manpages.debian.org/unstable/nftables/nft.8.en.html

ArchWiki Nftables page

https://wiki.archlinux.org/title/Nftables

и любым другим материалам, по *rfc* из *inet* .

ЛАБОРАТОРНАЯ РАБОТА № 16

Ограничение числа соединений. DDoS.

Цель работы. Применение сетевых технологий и утилит *Linux* для построения программных средств защиты от атак типа *DDoS* (отказа в обслуживании).

Последовательность выполнения работы:

1. Используйте технологию *iptables* и модуль *connlimit* для ограничения количества соединений к серверу с клиентских компьютеров (защита от DDoS-атак). Например, разрешение не более трех соединений по *ssh* на одного клиента реализуется правилом:

iptables -A INPUT -p tcp --syn --dport 22 -m connlimit
--connlimit-above 3 -j REJECT

2. Продемонстрируйте возможность установления ограничения на количество одновременных соединений на примере *ssh* сервера или используйте собственный сервер из предыдущих разделов лабораторных работ, по теме клиент-серверного взаимодействия на сокетах.

3. Исследуйте и продемонстрируйте и другие технологии ограничения числа соединений с целью защиты от DDoS атак (например, на уровне системного файервола Linux или ограничением количества открытых дескрипторов или ограничением на уровне приложений или с помощью инструмента DDoS Deflate и др.).

ЛАБОРАТОРНАЯ РАБОТА № 17

Nmap. Сканирование сети

Цель работы. Освоение утилиты, предназначенной для разнообразного настраиваемого сканирования IP-сетей с любым количеством объектов, для определения состояния этих объектов. *Nmap* использует множество различных методов сканирования, таких как *UDP*, *TCP* (*connect*), *TCP SYN* (полуоткрытое), *ftp-proxy* (прорыв через *ftp*), *ICMP* (*ping*), *FIN*, *ACK*, *SYN*- и *NULL*-сканирование и др

Последовательность выполнения работы:

1. Проведите аудит локальной сети сканером *Nmap* . Выбирайте в качестве объектов сканирования преимущественно компьютеры из своего сегмента локальной сети. Проверьте наличие открытых портов и протоколов, определите версию ОС на удаленном хосте, выполните по своему усмотрению 4–5 видов различного сканирования с помощью *Nmap* . Используйте всевозможные режимы сканирования для получения как можно более подробной информации о сегменте сети и о хостах, применяя различные опции, направляя вывод на консоль или в файл (с последующей фильтрацией), применяя временные параметры (расписания сканирования) и т.д.

2. Попытайтесь реализовать некоторые дополнительные возможности *Nmap*, : определение типа ОС удалённого хоста с использованием отпечатков

стека TCP/IP, ”невидимое” сканирование, динамическое вычисление времени задержки и повтор передачи пакетов, параллельное сканирование, определение неактивных хостов методом параллельного ping-опроса, сканирование с использованием ложных хостов, определение наличия пакетных фильтров, RPC-сканирование, сканирование с использованием IP-фрагментации, быстрый поиск уязвимостей SQL Injection и др.

3. По возможности примените скриптовый язык *Lua* - *Nmap Scripting Engine* (*NSE*) для написания соответствующих сценариев.

Информацию по функциональности *Nmap* можно найти, например, на ресурсах:

http://compress.ru/article.aspx?id=17371
https://www.youtube.com/watch?v=iUZ6nTMO8K0
https://www.youtube.com/watch?v=0xZqQDof-JA
https://www.youtube.com/watch?v=lXK5j2nRuv8
https://www.youtube.com/watch?v=kaJuZEW6D3I
https://www.youtube.com/watch?v=7CGvjshtkNk
https://www.youtube.com/watch?v=TyUtnOb-kS0

4. Сравните возможности сетевого сканера *Nmap* с другими известными средствами аудита сети.

ЛАБОРАТОРНАЯ РАБОТА № 18

Создание снифера на Socket L2/L3 level

Цель работы. Применение возможностей *RAW*-сокетов, предоставляющих доступ к полям заголовков сообщений протоколов уровней *L2* и *L3* модели *OSI*.

Последовательность выполнения работы:

1. Ознакомьтесь со спецификой применения "сырых" сокетов на канальном и на сетевом уровнях, например, на таких ресурсах, как:

https://www.opensourceforu.com/2015/03/a-guide-to-using-raw-sockets/

https://www.binarytides.com/raw-sockets-c-code-linux/

http://www.kernel.org/doc/man-pages/online/pages/man7/packet.7.html

https://sock-raw.org/papers/sock_raw

или, воспользуйтесь множеством других источников.

2. Разработайте и отладьте консольное приложение, обладающее возможностями снифера пакетов, используя технологию *RAW*-сокетов. Ориентируйтесь на сообщения протоколов *ARP*, *ICMP*, *UDP*, *TCP*.
Выбор протокола задается из командной строки при старте приложения.

3. Продемонстрируйте возможности перехвата сообщений отдельных протоколов с помощью разработанного приложения и сопоставьте с результатами работы других, общеизвестных сниферов.

ЛАБОРАТОРНАЯ РАБОТА № 19

Спуфер на RAW Socket L2/L3 level

Цель работы. Применение возможностей *RAW*-сокетов, предоставляющих доступ к полям заголовков сообщений протоколов уровней *L2* и *L3* модели *OSI*.

Последовательность выполнения работы:

1. Исследуйте возможности *RAW*-сокетов, предоставляющих доступ к полям заголовков сообщений протоколов уровней *L2* и *L3*.
Информацию по "сырым" сокетам можно найти, например, на таких ресурсах, как:

https://www.opensourceforu.com/2015/03/a-guide-to-using-raw-sockets/
https://www.binarytides.com/raw-sockets-c-code-linux/
http://www.kernel.org/doc/man-pages/online/pages/man7/packet.7.html
https://sock-raw.org/papers/sock_raw

или, воспользуйтесь множеством других источников.

2. Разработайте и отладьте консольное приложение, обладающее возможностями спуфера пакетов, используя технологию *RAW*-сокетов.
При отладке данного сетевого приложения используйте тестовые сегменты сети. Не допускайте исход модифицированных вашим приложением пакетов в публичную сеть. Для этого можно ограничиться формированием

сообщений протоколов не выше канального уровня или , например, *ARP*.

3. Продемонстрируйте возможности модификации полей заголовка, выбранного протокола с помощью разработанного приложения.

ЛАБОРАТОРНАЯ РАБОТА № 20

Шифрование средствами GnuPG

Цель работы. Применение утилиты *GnuPG* для шифрования сообщений.

Последовательность выполнения работы:

1. Инсталлируйте *GnuPG* (не ниже второй версии) на ваш компьютер:

sudo apt-get install gnupg2

Выведите на консоль информацию об установленной версии *GnuPG* и поддерживаемых криптоалгоритмах:

gpg2 --version

Набор основных команд можно увидеть по:

gpg2 –help

2. Приступайте к генерации пары (*private* и *public*) ключей:

gpg2 --gen-key

Выберите тип ключа 1 , чтобы была возможность и шифровать сообщения и подписывать их

Your selection? 1

RSA keys may be between 1024 and 4096 bits long.

What keysize do you want? (2048)

Можно выбрать *4096*, это будет надежнее, но генерация продлится дольше.

Please specify how long the key should be valid.

Выберите время жизни ключа, например, несколько недель *3w*.

GnuPG needs to construct a user ID to identify your key.

Введите ваш идентификатор, потом *GnuPG* еще попросит ввести ваш *email* и *Comment*. Это необходимо для идентификации ваших ключей.

3. *You need a Passphrase to protect your secret key*.

Введите пароль от закрытого (*private*) ключа и ЗАПОМНИТЕ его.

Теперь запустится процесс генерации ключевой пары, который будет длиться некоторое время, причем *GnuPG* будет вываливать просьбы:

We need to generate a lot of random bytes. It is a good idea to perform some other action (type on the keyboard, move the mouse, utilize the disks) during the prime generation; this gives the random number generator a better chance to gain enough entropy.

которые можно выполнять, что, в принципе, может ускорить процесс генерации ключей. Если, вдруг, появится сообщение

Not enough random bytes available. Please do some other work to give the OS a chance to collect more entropy! (Need 204 more bytes)

Тогда вам придется инсталлировать еще и демона для сбора энтропии:

sudo apt-get install rng-tools

4. По завершении процесса генерации появится информация подобная этой:

gpg: key 8640D6B9 marked as ultimately trusted

public and secret key created and signed.

gpg: checking the trustdb

gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model

gpg: depth: 0 valid: 1 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 1u

gpg: next trustdb check due at 2020-03-24

pub 4096R/8640D6B9 2020-02-25 [expires: 2020-03-24]

Key fingerprint = DB5E AA39 0745 427D ED31 D189 3197 3F00 8640 D6B9

uid Alexeev (March_24) <afiskon@example.com>

sub 4096R/5982B4BF 2020-02-25 [expires: 2020-03-24]

5. Получить *fingerprint* (отпечаток) ключа можно

gpg2 –fingerprint <e-mail>

Нужны *fingerprints* для того, чтобы в дальнейшем, проверять, что с сервера ключей был импортирован, действительно, правильный ключ.

Просмотреть список ключей можно командой

gpg2 –list-keys

Удалить ключи (если такое понадобится) можно, например :

gpg2 --delete-secret-keys 8640D6B9

gpg2 --delete-keys 8640D6B9

6. Зайдите в подкаталог *.gnupg* вашего *Home* каталога и проанализируйте его содержимое в результате генерации ключей.

Создайте в вашем *Home* каталоге рабочий подкаталог для упражнений с шифрованием файлов и подписями. Зайдите в него и создайте текстовый файл, например, *original.txt* с каким-либо содержимым для шифрования.

7. Выполните шифрование на вашем ключе:

gpg2 -a -r 8640D6B9 -e original.txt

Просмотрите содержимое созданного в результате файла

original.txt.asc, содержащего зашифрованную информацию:

cat original.txt.asc

Дешифрируйте текст с помощью

gpg2 -r 8640D6B9 -d original.txt.asc > decrypted.txt

You need a passphrase to unlock the secret key for

Введите пароль, закрытого (*private*) ключа (был на этапе генерации ключей).

Просмотрите содержимое созданного в результате файла *decrypted.txt* :

cat decrypted.txt

и убедитесь в его совпадении с исходным файлом *original.txt* .

8. Создайте цифровую подпись текстового файла

gpg2 -r 8640D6B9 --clearsign message.txt

Цифровая подпись попадает в файл *message.txt.asc* содержимое просматривается командой *cat message.txt.asc* .

Проверка цифровой подписи выполняется, как

gpg2 --verify message.txt.asc

Самостоятельно исследуйте режимы работы *GnuPG* с различными опциями и ключами и выполнение некоторых команд из базового списка.

ЛАБОРАТОРНАЯ РАБОТА № 21

Цифровая подпись, экспорт/импорт ключей

Цель работы. Создание электронных цифровых подписей (*ЭЦП*) различных типов. Экспорт (и импорт) ключей и их связок на ключевые сервера.

Последовательность выполнения работы:

1. Организуйте экспорт вашего открытого ключа на ключевой сервер, или в файл, для того, чтобы вам могли шифровать файлы/сообщения, а также проверять наши цифровые подписи.

Экспорт открытого ключа в текстовый файл:

gpg2 --export --armor 29FEB0EF > mykey.asc

Здесь *0x29FEB0EF* — отпечаток ключа вашей ключевой пары, открытый ключ которой экспортируем,

а mykey.asc — имя файла, в который будет сохранён результат.

2. Создание электронной цифровой подписи (*ЭЦП*) файла.

GnuPG позволяет использовать несколько типов подписей:

- *отсоединённая* в текстовом формате:
  создаётся файл с расширением **.asc* вида *mydocument.pdf.asc*
  (где *mydocument.pdf* — имя оригинального файла);
- *отсоединённая* в двоичном формате:
  создаётся файл с расширением **.sig* вида *mydocument.pdf.sig*
  в бинарном формате.
- *встроенная* в файл:
  содержимое файла изменяется так, чтобы в него была добавлена ЭЦП (в текстовом либо в двоичном формате). Чаще всего применяется при отправке подписанных сообщений по электронной почте;

Для создания *ЭЦП* файла используется *закрытый* ключ из нашей ключевой пары, а для проверки — *открытый*.

Создадим *отсоединённую ЭЦП* файла mydocument.pdf в текст. формате:

gpg2 --sign --detach-sign --default-key 29FEB0EF --armor mydocument.pdf

Создадим *отсоединённую* подпись в двоичном формате:

gpg2 --sign --detach-sign --default-key 29FEB0EF mydocument.pdf

на выходе будет получен файл *mydocument.pdf.sig*

Создадим *встроенную* в файл подпись в текстовом формате:

gpg2 --sign --default-key 29FEB0EF --armor mydocument.pdf

Создадим встроенную в файл подпись в двоичном формате:

gpg2 --sign --default-key 29FEB0EF mydocument.pdf

При создании встроенных подписей содержимое файла-источника целиком включается внутрь, поэтому использовать данный формат не желательно из-за дублирования и значительного размера. Поэтому *отсоединённая ЭЦП* является самым популярным вариантом подписи.

3. Импорт *открытого* ключа. Для проверки чужой цифровой подписи *GnuPG*, у нас должны быть:

- открытый ключ респондента (того человека, который её создал);
- оригинальный файл и файл отсоединённой цифровой подписи.

Сначала импортируем ключ респондента, подписавшего файл (если это не было сделано ранее). Это можно сделать любым способом:

- текстовый файл;
- серверы-хранилища ключей;
- буфер обмена (для GUI утилит).

Импортируем открытый ключ из файла:

gpg2 --import mykey.asc

Здесь *mykey.asc* — имя файла с открытым ключом.

Теперь мы должны установить доверие импортированному ключу, т.к. в противном случае не сможем проверить подпись.

Войдём в интерактивный режим:

pg2 --edit-key 29FEB0EF

Установим доверие ключу:

trust

Проверим отпечаток респондента (например посредством телефонного звонка или любым другим способом), затем выберем пункт

Я полностью доверяю (*I trust fully*).

Выходим из интерактивного режима:

quit

4. Проверка *ЭЦП*. Файл отсоединённой *ЭЦП* должен лежать в том же каталоге, что и оригинальный файл, иначе выполнить проверку его подлинности будет невозможно.

Проверка отсоединённой подписи файла:

gpg2 --verify mydocument.pdf.sig

5. Экспорт/импорт на ключевые сервера. Изучите вопросы экспорта (и импорта) ключей и их связок на ключевые сервера, реализуйте данные способы экспорта/импорта на какой-либо выбранный сервер, отобразив результаты в отчете.

6. Встраивание. Опробуйте возможности встраивания средств *GnuPG* в какую-либо среду коммуникаций, например, в почтовый клиент *Thunderbird* или др.

ЛАБОРАТОРНАЯ РАБОТА № 22

Обмен на утилитах netcat, cryptcat

Цель работы. Изучение утилит, позволяющих устанавливать соединения *TCP* и *UDP*, принимать и передавать данные, в открытом, а также в защищенном виде.

Последовательность выполнения работы:

1. Организуйте с помощью утилиты *Netcat* взаимодействие процессов на разных хостах (наподобие *pipe* , но только возможно на разных хостах) например, в виде чата, а также передачу файлов между хостами. Используйте также утилиту *Cryptcat* для шифрования трафика. Проанализируйте защищен-

ность трафика с помощью сетевого снифера *Wireshark* в случае открытого и защищенного соединений.

Сведения об утилитах *Netcat* и *Cryptcat* можно узнать, например, из ресурсов :

http://handynotes.ru/2010/01/unix-utility-netcat.html
https://www.youtube.com/watch?v=oNwLy7JTJl8
https://www.youtube.com/watch?v=Ro7VDzOU32Q
https://www.youtube.com/watch?v=z04YgdWx4_Y

ЛАБОРАТОРНАЯ РАБОТА № 23

Аудит сегмента Wi-Fi сети

Цель работы. Проведение аудита защищенности тестового сегмента сети *Wi-Fi*.

Последовательность выполнения работы:

1. Проанализируйте существующие средства, применяемые с целью аудита и выработки рекомендаций по безопасности сетей *Wi-Fi*.
2. Выберите какой-либо набор программных средств, из числа свободно распространяемых, например:

   airodump (снифер для сетей стандарта *802.11*),

   aireplay (для инжекции *Wi-Fi* фреймов),

   aircrack (взлом *WEP* и брутфорс *WPA-PSK*),

   airdecap (декодирование перехваченных *WEP/WPA* файлов).

   Набор средств может быть выбран и любым другим, платформа для установки и проведения исследования, также не лимитируется (*Win/Lin/Mac*).
3. С помощью выбранных средств оцените уязвимость тестовой отдельной сети *Wi-Fi*. Испытайте сети *Wi-Fi* с различными существующими типами защиты. Сформулируйте рекомендуемые меры по защите *Wi-Fi* сетей.

ЛАБОРАТОРНАЯ РАБОТА № 24

Системы обнаружения вторжений IDS

Цель работы. Сопоставление принципов действия систем обнаружения

вторжений (*Intrusion Detection Systems*). Получение навыков работы с выбранной *IDS*.

Последовательность выполнения работы:

1. Проанализируйте существующие системы обнаружения вторжений, отличающиеся принципом их действия:

- сигнатурные *IDS*,
- *IDS*, основанные на аномалиях,
- *IDS*, основанные на правилах .

Можно воспользоваться, например, этими ресурсами:

https://selectel.ru/blog/ips-and-ids/

https://habr.com/ru/company/ruvds/blog/506730/

https://habr.com/ru/company/ruvds/blog/507234/

https://cyvatar.ai/write-configure-snort-rules/

или любыми другими.

2. Выберите какой-либо набор программных средств, из числа свободно распространяемых (например, *Snort*, *Suricata*), выполните установку и конфигурацию. Продемонстрируйте основные режимы работы.

Сопоставьте выбранное решение с другими возможными.

ТРЕБОВАНИЯ К ОТЧЕТАМ

По результатам выполнения лабораторных работ составляется отчет. Рекомендуется составлять отчет параллельно с выполнением заданий лабораторных работ, не оставляя этот процесс «на потом». Отчет должен начинаться с титульного листа установленного образца, с названием университета, института высшей школы или кафедры. На титульном листе указывается ф.и.о. студента-исполнителя работы, номер студенческой группы и ф.и.о. преподавателя. Затем следует страница оглавления.

В отчете приводится цель работы и формулировки заданий (фрагменты текста заданий копируются из электронной версии данного учебного пособия), вслед за которыми помещается информация, подтверждающая выполнение студентом заданий лабораторной работы: скриншоты с результатами исполнения команд, программ, исходные тексты тех программ (с комментариями) или фрагментов, которые были составлены самим студентом, скрипты (*BASH* и др.) командных файлов, дампы памяти и пакетов. Отчет должен содержать ответы на поставленные в заданиях вопросы и, обязательно, выводы по работе. Отчеты по всем выполненным работам аккумулируются в едином документе. Описание очередной работы начинается с новой страницы отчета.

Защита лабораторных работ происходит по предъявлению, оформленного с учетом приведенных требований, отчета и сопровождается демонстрацией исполнения программ и сценариев, а также ответами на вопросы преподавателя.

Файл отчета предоставляется в формате .pdf и именуется фамилией и именем студента-исполнителя.

Библиографический список

1. Барабанова М. И. Информационные технологии: открытые системы, сети, безопасность в системах и сетях : учеб. пособие / М. И. Барабанова, В. И. Кияев. – СПб. : Изд-во СПбГУЭФ, 2010.

2. Олифер В., Олифер Н., Компьютерные сети. Принципы, технологии, протоколы : учебник для вузов / В.Олифер, Н.Олифер. – СПб. : Изд-во Питер, 2010.

3. Робачевский А. Операционная система UNIX / А. Робачевский. – СПб. : БХВ-Петербург, 1999.

4. Стивенс У. UNIX : взаимодействие процессов / У. Стивенс. – СПб. : Питер, 2002.

5. Тейнсли Д. Linux и UNIX : программирование в shell. Руководство разработчика / Д. Тейнсли. – Киев : Издательская группа BHV, 2001.

6. Чан Т. Системное программирование на C++ для UNIX / Т. Чан. – Киев : Издательская группа BHV, 1999.